Homework 2 - Drew Kearny - Due September 22

Name: Drew Kearny

netID: 231005935

Collaborated with:

Your homework **must be submitted in Word or PDF format, created by calling “Knit Word” or “Knit PDF” from RStudio on your R Markdown document. Submission in other formats may receive a grade of 0**. Your responses must be supported by both textual explanations and the code you generate to produce your result. Note that all R code used to produce your results must be shown in your knitted file. You can collaborate with your classmates, but you must identify their names above, and you must submit your own homework as a knitted file.

## Q1 Back to some R basics

Throughout, try to use as few lines of code as possible. Majority of tasks can be completed in one line.

1. Let’s start easy by working through some R basics, to continue to brush up on them. Define a variable x.vec to contain the integers 1 through 100. Check that it has length 100. Report the data type being stored in x.vec. Add up the numbers in x.vec, by calling one built-in R function.

x.vec <- 1:100  
  
length(x.vec) == 100

## [1] TRUE

class(x.vec)

## [1] "integer"

print(sum(x.vec))

## [1] 5050

1. Convert x.vec into a matrix with 20 rows and 5 columns, and store this as x.mat. Here x.mat should be filled out in the default order (column major order). Check the dimensions of x.mat, and the data type as well. Compute the sums of each of the 5 columns of x.mat, by calling a built-in R function. Check (using a comparison operator) that the sum of column sums of x.mat equals the sum of x.vec.

x.mat <- matrix(x.vec, nrow = 20, ncol = 5)  
  
dim(x.mat)

## [1] 20 5

class(x.mat)

## [1] "matrix" "array"

print(colSums(x.mat))

## [1] 210 610 1010 1410 1810

sum(colSums(x.mat)) == sum(x.mat)

## [1] TRUE

1. Extract and display rows 1, 5, and 17 of x.mat, with a single line of code. Answer the following questions, how many elements in row 2 of x.mat are larger than 40? How many elements in column 3 are in between 45 and 50 (exclusive)? How many elements in column 5 are odd? Hint: take advantage of the sum() function applied to Boolean vectors.

3 in row 2 are greater than 40, 4 in column 3 are between 45 and 50 exclusive, 10 in column 5 are odd.

x.mat[c(1, 5, 17), ]

## [,1] [,2] [,3] [,4] [,5]  
## [1,] 1 21 41 61 81  
## [2,] 5 25 45 65 85  
## [3,] 17 37 57 77 97

# row 2  
sum(x.mat[2, ] > 40)

## [1] 3

# col 3  
sum(x.mat[, 3] > 45 & x.mat[, 3] < 50)

## [1] 4

# col 10  
sum(x.mat[, 5] %% 2 == 1)

## [1] 10

1. Using Boolean indexing, modify x.vec so that every even number in this vector is incremented by 10, and every odd number is left alone. Print out the result to the console. Repeat using ifelse() to do the same thing, again using just a single line of code. Hint: the remainder of division by 2 for even numbers is 0.

new.x.vec <- x.vec  
  
new.x.vec[x.vec %% 2 == 0] <- new.x.vec[x.vec %% 2 == 0] + 10  
  
print(new.x.vec)

## [1] 1 12 3 14 5 16 7 18 9 20 11 22 13 24 15 26 17 28  
## [19] 19 30 21 32 23 34 25 36 27 38 29 40 31 42 33 44 35 46  
## [37] 37 48 39 50 41 52 43 54 45 56 47 58 49 60 51 62 53 64  
## [55] 55 66 57 68 59 70 61 72 63 74 65 76 67 78 69 80 71 82  
## [73] 73 84 75 86 77 88 79 90 81 92 83 94 85 96 87 98 89 100  
## [91] 91 102 93 104 95 106 97 108 99 110

# with ifelse()  
new.x.vec <- ifelse(x.vec %% 2 == 0, x.vec + 10, x.vec)  
print(new.x.vec)

## [1] 1 12 3 14 5 16 7 18 9 20 11 22 13 24 15 26 17 28  
## [19] 19 30 21 32 23 34 25 36 27 38 29 40 31 42 33 44 35 46  
## [37] 37 48 39 50 41 52 43 54 45 56 47 58 49 60 51 62 53 64  
## [55] 55 66 57 68 59 70 61 72 63 74 65 76 67 78 69 80 71 82  
## [73] 73 84 75 86 77 88 79 90 81 92 83 94 85 96 87 98 89 100  
## [91] 91 102 93 104 95 106 97 108 99 110

1. Consider the list x.list created below. Complete the following tasks, each with a single line of code: extract all but the second element of x.list—seeking here a list as the final answer. Extract the first and third elements of x.list, then extract the second element of the resulting list—seeking here a vector as the final answer. Extract the second element of x.list as a vector, and then extract the first 10 elements of this vector—seeking here a vector as the final answer. Note: pay close attention to what is asked and use either single brackets [ ] or double brackets [[ ]] as appropriate.

x.list = list(rnorm(6), letters, sample(c(TRUE,FALSE), size=4, replace=TRUE))

#list  
x.list[-2]

## [[1]]  
## [1] 1.4495528 -1.2250530 0.1593396 -0.6667938 -0.3180395 -1.2352150  
##   
## [[2]]  
## [1] TRUE TRUE FALSE TRUE

first.third <- x.list[c(1, 3)]  
# list  
first.third

## [[1]]  
## [1] 1.4495528 -1.2250530 0.1593396 -0.6667938 -0.3180395 -1.2352150  
##   
## [[2]]  
## [1] TRUE TRUE FALSE TRUE

#vector  
first.third[[2]]

## [1] TRUE TRUE FALSE TRUE

second <- x.list[[2]]  
#list  
second

## [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"  
## [20] "t" "u" "v" "w" "x" "y" "z"

#vector  
second[1:10]

## [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"

## Q2 Basic data frame manipulations

Below we construct a data frame, of 50 states x 10 variables. The first 8 variables are numeric and the last 2 are factors. The numeric variables here come from the built-in state.x77 matrix, which records various demographic factors on 50 US states, measured in the 1970s. You can learn more about this state data set by typing ?state.x77 into your R console.

state.df = data.frame(state.x77, Region=state.region, Division=state.division)

1. Add a column to state.df, containing the state abbreviations that are stored in the built-in vector state.abb. Name this column Abbr. You can do this in (at least) two ways: by using a call to data.frame(), or by directly defining state.df$Abbr. Display the first 3 rows and all 11 columns of the new state.df.

state.df$Abbr <- state.abb  
  
  
head(state.df, n = 3)

## Population Income Illiteracy Life.Exp Murder HS.Grad Frost Area  
## Alabama 3615 3624 2.1 69.05 15.1 41.3 20 50708  
## Alaska 365 6315 1.5 69.31 11.3 66.7 152 566432  
## Arizona 2212 4530 1.8 70.55 7.8 58.1 15 113417  
## Region Division Abbr  
## Alabama South East South Central AL  
## Alaska West Pacific AK  
## Arizona West Mountain AZ

1. Remove the Region column from state.df. You can do this in (at least) two ways: by using negative indexing, or by directly setting state.df$Region to be NULL. Display the first 3 rows and all 10 columns of state.df.

state.df$Region <- NULL  
  
head(state.df, n = 3)

## Population Income Illiteracy Life.Exp Murder HS.Grad Frost Area  
## Alabama 3615 3624 2.1 69.05 15.1 41.3 20 50708  
## Alaska 365 6315 1.5 69.31 11.3 66.7 152 566432  
## Arizona 2212 4530 1.8 70.55 7.8 58.1 15 113417  
## Division Abbr  
## Alabama East South Central AL  
## Alaska Pacific AK  
## Arizona Mountain AZ

1. Add two columns to state.df, containing the x and y coordinates (longitude and latitude, respectively) of the center of the states, that are stored in the (existing) list state.center. Hint: take a look at this list in the console, to see what its elements are named. Name these two columns Center.x and Center.y. Display the first 3 rows and all 12 columns of state.df.

state.df$Center.x <- state.center$x  
state.df$Center.y <- state.center$y  
  
head(state.df, n = 3)

## Population Income Illiteracy Life.Exp Murder HS.Grad Frost Area  
## Alabama 3615 3624 2.1 69.05 15.1 41.3 20 50708  
## Alaska 365 6315 1.5 69.31 11.3 66.7 152 566432  
## Arizona 2212 4530 1.8 70.55 7.8 58.1 15 113417  
## Division Abbr Center.x Center.y  
## Alabama East South Central AL -86.7509 32.5901  
## Alaska Pacific AK -127.2500 49.2500  
## Arizona Mountain AZ -111.6250 34.2192

1. Make a new data frame which contains only those states whose longitude is less than -100. Do this in two different ways: using manual indexing called state.sub1, and subset() called state.sub2. Check that they are equal to each other, using an appropriate function call.

state.sub1 <- state.df[state.df$Center.x < -100, ]  
  
state.sub2 <- subset(state.df, Center.x < -100)  
  
identical(state.sub1, state.sub2)

## [1] TRUE

1. Make a new data frame, state.sub3, which contains only the states whose longitude is less than -100, and whose murder rate is above 9%. Print this new data frame to the console. Among the states in this new data frame, which has the highest average life expectancy?

state.sub3 <- subset(state.df, Center.x < -100 & Murder > 9)  
  
print(state.sub3)

## Population Income Illiteracy Life.Exp Murder HS.Grad Frost Area  
## Alaska 365 6315 1.5 69.31 11.3 66.7 152 566432  
## California 21198 5114 1.1 71.71 10.3 62.6 20 156361  
## Nevada 590 5149 0.5 69.03 11.5 65.2 188 109889  
## New Mexico 1144 3601 2.2 70.32 9.7 55.2 120 121412  
## Division Abbr Center.x Center.y  
## Alaska Pacific AK -127.250 49.2500  
## California Pacific CA -119.773 36.5341  
## Nevada Mountain NV -116.851 39.1063  
## New Mexico Mountain NM -105.942 34.4764

state.sub3[which.max(state.sub3$Life.Exp), ]

## Population Income Illiteracy Life.Exp Murder HS.Grad Frost Area  
## California 21198 5114 1.1 71.71 10.3 62.6 20 156361  
## Division Abbr Center.x Center.y  
## California Pacific CA -119.773 36.5341

## Q3 Practice with the apply family

We’re going to look at a data set on 97 men who have prostate cancer (from the book [The Elements of Statistical Learning](http://statweb.stanford.edu/~hastie/ElemStatLearn/)). There are 9 variables measured on these 97 men:

1. lpsa: log PSA score
2. lcavol: log cancer volume
3. lweight: log prostate weight
4. age: age of patient
5. lbph: log of the amount of benign prostatic hyperplasia
6. svi: seminal vesicle invasion
7. lcp: log of capsular penetration
8. gleason: Gleason score
9. pgg45: percent of Gleason scores 4 or 5

Use the following to load the prostate cancer data set into your R session, and store it as a data frame pros.df:

pros.dat <- read.table("http://www.stat.cmu.edu/~ryantibs/statcomp/data/pros.dat")

1. Using sapply(), calculate the mean of each variable. Also, calculate the standard deviation of each variable. Each should require just one line of code. Display your results.

# Create the data frame  
pros.df <- data.frame(pros.dat)  
  
print(sapply(pros.df, mean))

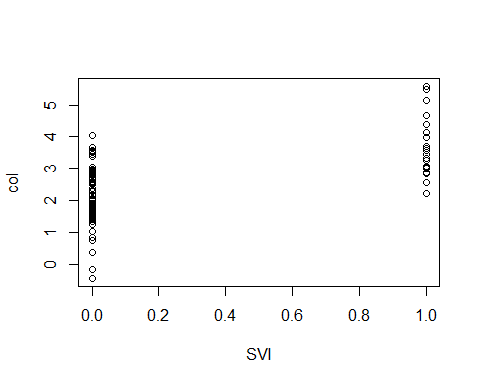
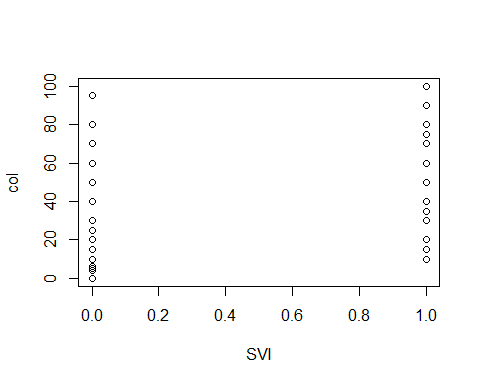
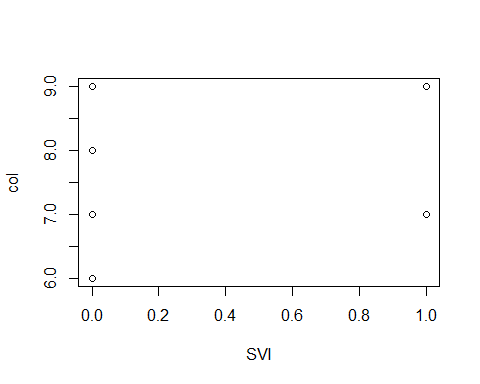
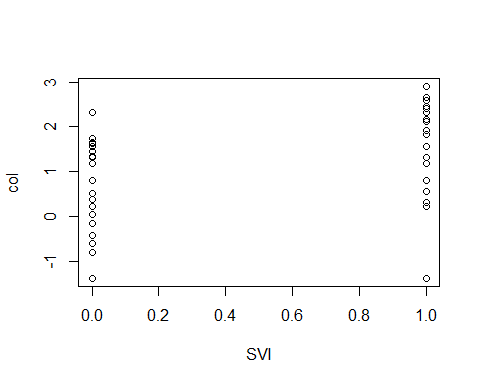
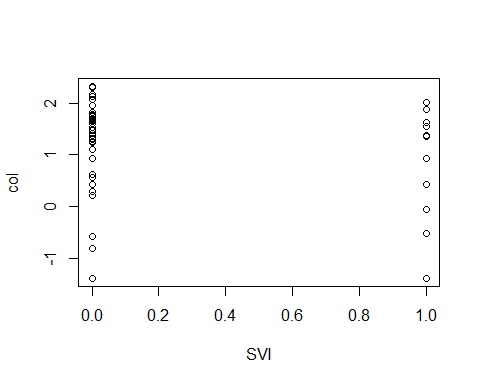
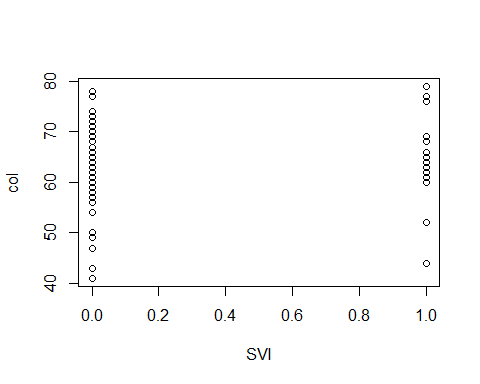
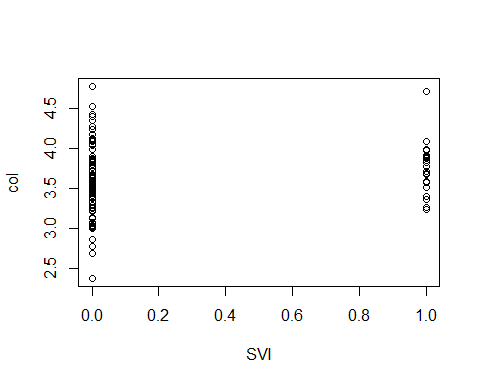
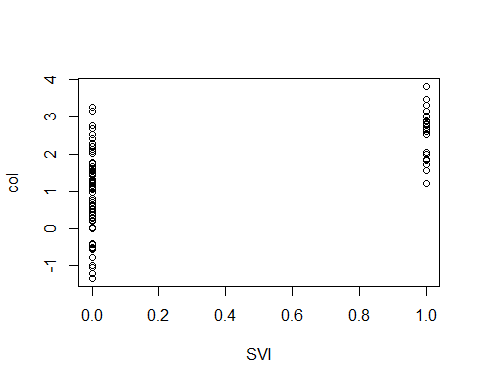
## lcavol lweight age lbph svi lcp gleason   
## 1.3500096 3.6289427 63.8659794 0.1003556 0.2164948 -0.1793656 6.7525773   
## pgg45 lpsa   
## 24.3814433 2.4783869

print(sapply(pros.df, sd))

## lcavol lweight age lbph svi lcp gleason   
## 1.1786249 0.4284112 7.4451171 1.4508066 0.4139949 1.3982496 0.7221341   
## pgg45 lpsa   
## 28.2040346 1.1543291

1. Let’s plot each variable against SVI. Using lapply(), plot each column, excluding SVI, on the y-axis with SVI on the x-axis. This should require just one line of code.

lapply(pros.df[, -5], function(col) plot(pros.df$svi, col, xlab = "SVI", ylab = names(col)))



## $lcavol  
## NULL  
##   
## $lweight  
## NULL  
##   
## $age  
## NULL  
##   
## $lbph  
## NULL  
##   
## $lcp  
## NULL  
##   
## $gleason  
## NULL  
##   
## $pgg45  
## NULL  
##   
## $lpsa  
## NULL

1. Now, use lapply() to perform t-tests for each variable in the data set, between SVI and non-SVI groups. To be precise, you will perform a t-test for each variable excluding the SVI variable itself. For convenience, we’ve defined a function t.test.by.ind() below, which takes a numeric variable x, and then an indicator variable ind (of 0s and 1s) that defines the groups. Run this function on the columns of pros.dat, excluding the SVI column itself, and save the result as tests. What kind of data structure is tests? Print it to the console.

The Data Structure of tests is a list

t.test.by.ind = function(x, ind) {  
 stopifnot(all(ind %in% c(0, 1)))  
 return(t.test(x[ind == 0], x[ind == 1]))  
}

tests <- lapply(X = pros.dat[, -5], FUN = t.test.by.ind, ind = pros.dat$svi)  
  
  
print(tests)

## $lcavol  
##   
## Welch Two Sample t-test  
##   
## data: x[ind == 0] and x[ind == 1]  
## t = -8.0351, df = 51.172, p-value = 1.251e-10  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -1.917326 -1.150810  
## sample estimates:  
## mean of x mean of y   
## 1.017892 2.551959   
##   
##   
## $lweight  
##   
## Welch Two Sample t-test  
##   
## data: x[ind == 0] and x[ind == 1]  
## t = -1.8266, df = 42.949, p-value = 0.07472  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -0.33833495 0.01674335  
## sample estimates:  
## mean of x mean of y   
## 3.594131 3.754927   
##   
##   
## $age  
##   
## Welch Two Sample t-test  
##   
## data: x[ind == 0] and x[ind == 1]  
## t = -1.1069, df = 30.212, p-value = 0.2771  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -6.018547 1.786718  
## sample estimates:  
## mean of x mean of y   
## 63.40789 65.52381   
##   
##   
## $lbph  
##   
## Welch Two Sample t-test  
##   
## data: x[ind == 0] and x[ind == 1]  
## t = 0.88281, df = 34.337, p-value = 0.3835  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -0.3914341 0.9930934  
## sample estimates:  
## mean of x mean of y   
## 0.1654837 -0.1353460   
##   
##   
## $lcp  
##   
## Welch Two Sample t-test  
##   
## data: x[ind == 0] and x[ind == 1]  
## t = -8.8355, df = 31.754, p-value = 4.58e-10  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -2.797674 -1.749133  
## sample estimates:  
## mean of x mean of y   
## -0.6715458 1.6018579   
##   
##   
## $gleason  
##   
## Welch Two Sample t-test  
##   
## data: x[ind == 0] and x[ind == 1]  
## t = -3.6194, df = 36.843, p-value = 0.0008816  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -0.8718223 -0.2459721  
## sample estimates:  
## mean of x mean of y   
## 6.631579 7.190476   
##   
##   
## $pgg45  
##   
## Welch Two Sample t-test  
##   
## data: x[ind == 0] and x[ind == 1]  
## t = -4.9418, df = 31.288, p-value = 2.482e-05  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -44.04052 -18.31537  
## sample estimates:  
## mean of x mean of y   
## 17.63158 48.80952   
##   
##   
## $lpsa  
##   
## Welch Two Sample t-test  
##   
## data: x[ind == 0] and x[ind == 1]  
## t = -6.8578, df = 33.027, p-value = 7.879e-08  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -2.047129 -1.110409  
## sample estimates:  
## mean of x mean of y   
## 2.136592 3.715360

class(tests)

## [1] "list"

1. Using lapply() again, extract the p-values from the tests object you created in the last question, with just a single line of code. Hint: first, take a look at the first element of tests, what kind of object is it, and how is the p-value stored? Second, run the command "[["(pros.df, "lcavol") in your console—what does this do? It is similar to the concept in the figure for lapply() in the notes. Now use what you’ve learned to extract p-values from the tests object.

lapply(tests, "[[", "p.value")

## $lcavol  
## [1] 1.25104e-10  
##   
## $lweight  
## [1] 0.07472088  
##   
## $age  
## [1] 0.2770533  
##   
## $lbph  
## [1] 0.3834772  
##   
## $lcp  
## [1] 4.579752e-10  
##   
## $gleason  
## [1] 0.0008816293  
##   
## $pgg45  
## [1] 2.482255e-05  
##   
## $lpsa  
## [1] 7.879066e-08

## Q4 Basic indexing and calculations

Use the following to load the prostate cancer data set into your R session, and store it as a matrix pros.dat:

pros.dat =  
 as.matrix(read.table("http://www.stat.cmu.edu/~ryantibs/statcomp/data/pros.dat"))

1. What are the dimensions of pros.dat (i.e., how many rows and how many columns)? Using integer indexing, print the first 6 rows and all columns; again using integer indexing, print the last 6 rows and all columns.

pros.dat has 97 rows and 9 columns

dim(pros.dat)

## [1] 97 9

print(pros.dat[1:6, ])

## lcavol lweight age lbph svi lcp gleason pgg45 lpsa  
## 1 -0.5798185 2.769459 50 -1.386294 0 -1.386294 6 0 -0.4307829  
## 2 -0.9942523 3.319626 58 -1.386294 0 -1.386294 6 0 -0.1625189  
## 3 -0.5108256 2.691243 74 -1.386294 0 -1.386294 7 20 -0.1625189  
## 4 -1.2039728 3.282789 58 -1.386294 0 -1.386294 6 0 -0.1625189  
## 5 0.7514161 3.432373 62 -1.386294 0 -1.386294 6 0 0.3715636  
## 6 -1.0498221 3.228826 50 -1.386294 0 -1.386294 6 0 0.7654678

print(pros.dat[(nrow(pros.dat) - 5):nrow(pros.dat), ])

## lcavol lweight age lbph svi lcp gleason pgg45 lpsa  
## 92 2.532903 3.677566 61 1.3480732 1 -1.386294 7 15 4.129551  
## 93 2.830268 3.876396 68 -1.3862944 1 1.321756 7 60 4.385147  
## 94 3.821004 3.896909 44 -1.3862944 1 2.169054 7 40 4.684443  
## 95 2.907447 3.396185 52 -1.3862944 1 2.463853 7 10 5.143124  
## 96 2.882564 3.773910 68 1.5581446 1 1.558145 7 80 5.477509  
## 97 3.471966 3.974998 68 0.4382549 1 2.904165 7 20 5.582932

1. Using the built-in R functions head() and tail() (i.e., do *not* use integer indexing), print the first 6 rows and all columns, and also the last 6 rows and all columns.

print(head(pros.dat, n = 6))

## lcavol lweight age lbph svi lcp gleason pgg45 lpsa  
## 1 -0.5798185 2.769459 50 -1.386294 0 -1.386294 6 0 -0.4307829  
## 2 -0.9942523 3.319626 58 -1.386294 0 -1.386294 6 0 -0.1625189  
## 3 -0.5108256 2.691243 74 -1.386294 0 -1.386294 7 20 -0.1625189  
## 4 -1.2039728 3.282789 58 -1.386294 0 -1.386294 6 0 -0.1625189  
## 5 0.7514161 3.432373 62 -1.386294 0 -1.386294 6 0 0.3715636  
## 6 -1.0498221 3.228826 50 -1.386294 0 -1.386294 6 0 0.7654678

print(tail(pros.dat, n = 6))

## lcavol lweight age lbph svi lcp gleason pgg45 lpsa  
## 92 2.532903 3.677566 61 1.3480732 1 -1.386294 7 15 4.129551  
## 93 2.830268 3.876396 68 -1.3862944 1 1.321756 7 60 4.385147  
## 94 3.821004 3.896909 44 -1.3862944 1 2.169054 7 40 4.684443  
## 95 2.907447 3.396185 52 -1.3862944 1 2.463853 7 10 5.143124  
## 96 2.882564 3.773910 68 1.5581446 1 1.558145 7 80 5.477509  
## 97 3.471966 3.974998 68 0.4382549 1 2.904165 7 20 5.582932

1. Does the matrix pros.dat have names assigned to its rows and columns, and if so, what are they? Use rownames() and colnames() to find out. Note: these would have been automatically created by the read.table() function that we used above to read the data file into our R session.

No row namews just numbers, but it does have column names.

rownames(pros.dat)

## [1] "1" "2" "3" "4" "5" "6" "7" "8" "9" "10" "11" "12" "13" "14" "15"  
## [16] "16" "17" "18" "19" "20" "21" "22" "23" "24" "25" "26" "27" "28" "29" "30"  
## [31] "31" "32" "33" "34" "35" "36" "37" "38" "39" "40" "41" "42" "43" "44" "45"  
## [46] "46" "47" "48" "49" "50" "51" "52" "53" "54" "55" "56" "57" "58" "59" "60"  
## [61] "61" "62" "63" "64" "65" "66" "67" "68" "69" "70" "71" "72" "73" "74" "75"  
## [76] "76" "77" "78" "79" "80" "81" "82" "83" "84" "85" "86" "87" "88" "89" "90"  
## [91] "91" "92" "93" "94" "95" "96" "97"

colnames(pros.dat)

## [1] "lcavol" "lweight" "age" "lbph" "svi" "lcp" "gleason"  
## [8] "pgg45" "lpsa"

1. Using named indexing, pull out the two columns of pros.dat that measure the log cancer volume and the log cancer weight, and store the result as a matrix pros.dat.sub. (Recall the explanation of variables at the top of this lab.) Check that its dimensions make sense to you, and that its first 6 rows are what you’d expect. Did R automatically assign column names to pros.dat.sub?

Everything is what I would expect and R did automatically assign column names.

pros.dat.sub <- pros.dat[, c("lcavol", "lweight")]  
  
dim(pros.dat.sub)

## [1] 97 2

print(head(pros.dat.sub, n = 6))

## lcavol lweight  
## 1 -0.5798185 2.769459  
## 2 -0.9942523 3.319626  
## 3 -0.5108256 2.691243  
## 4 -1.2039728 3.282789  
## 5 0.7514161 3.432373  
## 6 -1.0498221 3.228826

print(colnames(pros.dat.sub))

## [1] "lcavol" "lweight"

1. Using the log cancer weights and log cancer volumes, calculate the log cancer density for the 97 men in the data set.

There are multiple ways to accomplish this calculation. You should be able to perform this computation for all 97 men *with a single line of code*, taking advantage of R’s ability to vectorize.

pros.dat[, "lweight"] - pros.dat[, "lcavol"]

## 1 2 3 4 5 6 7   
## 3.34927749 4.31387827 3.20206862 4.48676180 2.68095691 4.27864812 2.73635393   
## 8 9 10 11 12 13 14   
## 2.84636182 4.31603779 3.02140045 3.34949578 4.94575465 1.40943107 1.52118028   
## 15 16 17 18 19 20 21   
## 2.23604819 1.51989293 3.93152844 1.36087283 3.82978492 3.64305344 2.27196255   
## 22 23 24 25 26 27 28   
## 1.44180417 3.92060717 1.66986487 3.28213760 1.67764602 3.20682737 4.26645657   
## 29 30 31 32 33 34 35   
## 2.08867429 0.96623583 3.80499006 3.62211644 1.76199120 3.25771567 3.22692434   
## 36 37 38 39 40 41 42   
## 2.81151718 2.23402267 1.91748115 1.42417741 2.21557380 2.52141851 2.24040801   
## 43 44 45 46 47 48 49   
## 3.28376338 2.12535224 1.92335630 1.72890290 1.26759217 2.87197419 1.75230647   
## 50 51 52 53 54 55 56   
## 2.34729308 2.90167970 2.57469997 3.12080737 1.99443248 0.36242264 3.01318440   
## 57 58 59 60 61 62 63   
## 1.89049436 3.30094798 3.63590171 2.78995450 4.06707715 1.72223329 0.74918015   
## 64 65 66 67 68 69 70   
## 1.88230535 1.54983507 2.37760598 1.85559481 1.85257993 4.85483410 3.58646053   
## 71 72 73 74 75 76 77   
## 1.72911387 2.18107208 2.61046226 1.39775493 0.84985684 0.12271852 2.42289400   
## 78 79 80 81 82 83 84   
## 1.81712679 0.93382880 1.04375180 1.60250165 0.95986194 1.27574735 1.16078501   
## 85 86 87 88 89 90 91   
## 2.14756070 0.21613074 1.70750593 1.63736246 1.91045817 2.13276370 0.85532601   
## 92 93 94 95 96 97   
## 1.14466315 1.04612817 0.07590539 0.48873764 0.89134643 0.50303155

1. Append the log cancer density to the columns of pros.dat, using cbind(). The new pros.dat matrix should now have 10 columns. Set the last column name to be ldens. Print its first 6 rows, to check that you’ve done all this right.

ldens <- pros.dat[, "lweight"] - pros.dat[, "lcavol"]  
pros.dat <- cbind(pros.dat, ldens)  
print(head(pros.dat, n = 6))

## lcavol lweight age lbph svi lcp gleason pgg45 lpsa  
## 1 -0.5798185 2.769459 50 -1.386294 0 -1.386294 6 0 -0.4307829  
## 2 -0.9942523 3.319626 58 -1.386294 0 -1.386294 6 0 -0.1625189  
## 3 -0.5108256 2.691243 74 -1.386294 0 -1.386294 7 20 -0.1625189  
## 4 -1.2039728 3.282789 58 -1.386294 0 -1.386294 6 0 -0.1625189  
## 5 0.7514161 3.432373 62 -1.386294 0 -1.386294 6 0 0.3715636  
## 6 -1.0498221 3.228826 50 -1.386294 0 -1.386294 6 0 0.7654678  
## ldens  
## 1 3.349277  
## 2 4.313878  
## 3 3.202069  
## 4 4.486762  
## 5 2.680957  
## 6 4.278648

## Q5 Computing standard deviations using iteration

Let’s reload the prostate data so the prompts do not depend on the previous question.

pros.dat =  
 as.matrix(read.table("http://www.stat.cmu.edu/~ryantibs/statcomp/data/pros.dat"))

We will also recreate the pros.dat.svi and pros.dat.no.svi from Lab.

pros.dat.svi <- pros.dat[pros.dat[,"svi"]==1,]  
pros.dat.no.svi <- pros.dat[pros.dat[,"svi"]==0,]

1. Take a look at the starter code below. The first line defines an empty vector pros.dat.svi.sd of length ncol(pros.dat) (of length 9). The second line defines an index variable i and sets it equal to 1. Write a third line of code to compute the standard deviation of the ith column of pros.dat.svi, using a built-in R function, and store this value in the ith element of pros.dat.svi.sd.

pros.dat.svi.sd = vector(length=ncol(pros.dat.svi))  
i = 1

pros.dat.svi.sd[i] <- sd(pros.dat.svi[, i])

1. Repeat the calculation as in the previous question, but for patients without SVI. That is, produce three lines of code: the first should define an empty vector pros.dat.no.svi.sd of length ncol(pros.dat) (of length 9), the second should define an index variable i and set it equal to 1, and the third should fill the ith element of pros.dat.no.svi.sd with the standard deviation of the ith column of pros.dat.no.svi.

pros.dat.no.svi.sd <- vector(length = ncol(pros.dat.no.svi))  
i <- 1  
pros.dat.no.svi.sd[i] <- sd(pros.dat.no.svi[, i])

1. Write a for() loop to compute the standard deviations of the columns of pros.dat.svi and pros.dat.no.svi, and store the results in the vectors pros.dat.svi.sd and pros.dat.no.svi.sd, respectively, that were created above. Note: you should have a single for() loop here, not two for loops. And if it helps, consider breaking this task down into two steps: as the first step, write a for() loop that iterates an index variable i over the integers between 1 and the number of columns of pros.dat (don’t just manually write 9 here, pull out the number of columns programmatically), with an empty body. As the second step, paste relevant pieces of your solution code from part a and b into the body of the for() loop. Print out the resulting vectors pros.dat.svi.sd and pros.dat.no.svi.sd to the console.

for (i in 1:ncol(pros.dat)) {  
 pros.dat.svi.sd[i] <- sd(pros.dat.svi[, i])  
 pros.dat.no.svi.sd[i] <- sd(pros.dat.no.svi[, i])  
}  
  
print(pros.dat.svi.sd)

## [1] 0.6707867 0.3275689 7.8715885 1.3545258 0.0000000 1.0452899 0.6015852  
## [8] 25.7344498 0.9251229

print(pros.dat.no.svi.sd)

## [1] 1.0685730 0.4479291 7.3105907 1.4782007 0.0000000 1.0379398 0.7088414  
## [8] 25.0667600 0.9646403

## Q6 Computing t-tests using vectorization

Use the pros.dat matrix from the read.table() function for the following parts.

1. Recall that the **two-sample (unpaired) t-statistic** between data sets and is:

* where is the sample mean of , is the sample variance of , and similarly for and . We will compute these t-statistics for all 9 numeric variables in our prostate data set, where will play the role of one of the variables for SVI patients, and will play the role of this variable for non-SVI patients. Start by computing a vector of the denominators of the t-statistics, called pros.dat.denom, according to the formula above. Take advantage of vectorization. Make sure not to include any hard constants (e.g., don’t just manually write 21 here for ); as always, programmatically define all the relevant quantities. Then compute a vector of t-statistics for the 9 variables in our data set, called pros.dat.t.stat, according to the formula above, and using pros.dat.denom. Again, take advantage of vectorization; this calculation should require just a single line of code. Print out the t-statistics to the console.

mean.X <- colMeans(pros.dat.svi)  
mean.Y <- colMeans(pros.dat.no.svi)  
var.X <- apply(pros.dat.svi, 2, var)  
var.Y <- apply(pros.dat.no.svi, 2, var)  
size.X <- nrow(pros.dat.svi)  
size.Y <- nrow(pros.dat.no.svi)  
# denominators  
pros.dat.denom <- sqrt((var.X / size.X) + (var.Y / size.Y))  
  
# T-stat   
pros.dat.t.stat <- (mean.X - mean.Y) / pros.dat.denom  
print(pros.dat.t.stat)

## lcavol lweight age lbph svi lcp gleason   
## 8.0351008 1.8265652 1.1069463 -0.8828113 Inf 8.8354810 3.6193860   
## pgg45 lpsa   
## 4.9417916 6.8578171

1. Given data and and the t-statistic as defined the last question, the **degrees of freedom** associated with is:

* Compute the degrees of freedom associated with each of our 9 t-statistics (from our 9 variables), storing the result in a vector called pros.dat.df. This might look like a complicated/ugly calculation, but really, it’s not too bad: it only involves arithmetic operators, and taking advantage of vectorization, the calculation should only require a single line of code. Hint: to simplify this line of code, it will help to first set short variable names for variables/quantities you will be using, as in sx = pros.dat.svi.sd, n = nrow(pros.dat.svi), and so on. Print out these degrees of freedom values to the console.

sx <- pros.dat.svi.sd  
sy <- pros.dat.no.svi.sd  
nx <- nrow(pros.dat.svi)  
ny <- nrow(pros.dat.no.svi)  
  
pros.dat.df <- ((sx^2 / nx + sy^2 / ny)^2) / (((sx^2 / nx)^2 / (nx - 1)) + ((sy^2 / ny)^2 / (ny - 1)))  
print(pros.dat.df)

## [1] 51.17241 42.94873 30.21179 34.33734 NaN 31.75374 36.84259 31.28770  
## [9] 33.02725

The function pt() evaluates the distribution function of the t-distribution. E.g.,

pt(x, df=v, lower.tail=FALSE)

returns the probability that a t-distributed random variable, with v degrees of freedom, exceeds the value x. Importantly, pt() is vectorized: if x is a vector, and so is v, then the above returns, in vector format: the probability that a t-distributed variate with v[1] degrees of freedom exceeds x[1], the probability that a t-distributed variate with v[2] degrees of freedom exceeds x[2], and so on.

1. Call pt() as in the above line, but replace x by the absolute values of the t-statistics you computed for the 9 variables in our data set, and v by the degrees of freedom values associated with these t-statistics. Multiply the output by 2, and store it as a vector pros.dat.p.val. These are called **p-values** for the t-tests of mean difference between SVI and non-SVI patients, over the 9 variables in our data set. Print out the p-values to the console. Identify the variables for which the p-value is smaller than 0.05 (hence deemed to have a significant difference between SVI and non-SVI patients). Identify the variable with the smallest p-value (the most significant difference between SVI and non-SVI patients).

pros.dat.p.val <- 2 \* (1 - pt(abs(pros.dat.t.stat), df = pros.dat.df))  
print(pros.dat.p.val)

## lcavol lweight age lbph svi lcp   
## 1.251039e-10 7.472088e-02 2.770533e-01 3.834772e-01 NaN 4.579752e-10   
## gleason pgg45 lpsa   
## 8.816293e-04 2.482255e-05 7.879066e-08

significant.pval <- which(pros.dat.p.val < 0.05)  
print(significant.pval)

## lcavol lcp gleason pgg45 lpsa   
## 1 6 7 8 9

smallest.pval <- which.min(pros.dat.p.val)  
print(smallest.pval)

## lcavol   
## 1